PLFS: A Checkpoint Filesystem for Parallel Applications

John Bent, HPC-5; Garth Gibson, Carnegie Mellon University; Gary Grider, HPC-DO;
Ben McClelland, HPC-3; Paul Nowoczynski, Pittsburgh Supercomputing Center;
James Nunez, Milo Polte, Meghan Wingate, HPC-5

Given the scale of massively parallel systems, system failures are not a matter of if, but of when. When an inevitable system failure takes place, it is not viable to restart the calculation from the beginning, particularly if such an application (e.g., a complex simulation) has been running for a month or so. Thus, massively parallel applications rely on checkpointing, which consists of saving snapshots of the current state of an application into files on persistent parallel storage systems. After a failure, users can restart the application from the most recent snapshot.

At present, concurrent and random-access checkpoint writes to a shared file require unnecessary 1) disk-seeking, and 2) file-locking at the parallel storage system. Such requirements drastically reduce bandwidth by as much as two orders of magnitude. To overcome this problem, we developed software known as Parallel Log-structured File System (PLFS), which decouples concurrent access and reorganizes logical writes into sequential physical writes, thus enabling applications to write in parallel at a near-optimal storage bandwidth.

PLFS works as follows: For every logical PLFS file created, PLFS creates a hierarchical directory structure, called a PLFS container, on the underlying parallel storage system. Hidden from users, this container consists of a single top-level directory and multiple subdirectories that contain the actual file data. To build a logical view of the file,

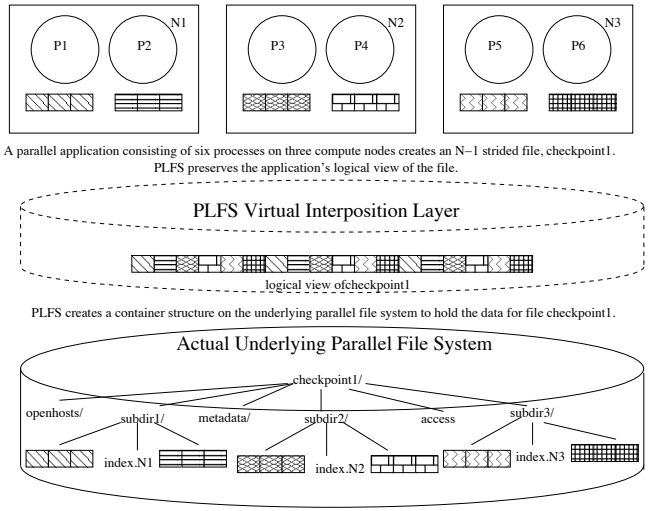


Fig. 1. How PLFS rewrites a checkpoint file.

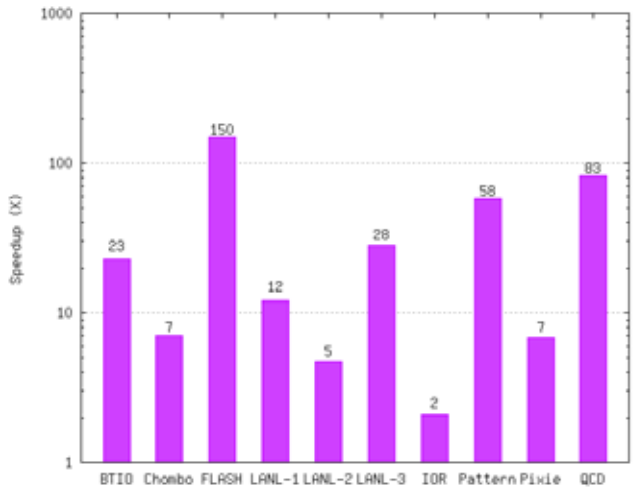


Fig. 2. These bars show the speedup in write bandwidth for eight different applications and two synthetics benchmarks. Note that the y-axis is logscale.

PLFS uses the data and metadata within the container to implement a virtual interposition, or intermediate, layer. Multiple processes opening the same logical file for writing share the container, although each opening receives a unique data file within the container into which all of its writes are appended. By giving each writing process in a parallel application access to a nonshared data file, PLFS transparently converts write-access patterns to improve checkpoint bandwidth by as much as several orders of magnitude.

Figure 1 shows how PLFS reorganizes a N-1 checkpoint file onto an underlying parallel system. The processes create a new file on PLFS called checkpoint1, causing PLFS in turn to create a contained structure on the underlying parallel file system. The figure also shows the access file, which is used to store ownership and privileged information about the logical file, the openhosts, and the metadata directories, all of which are use to cache metadata to improve query time.

File systems have two basic checkpointing patterns: N-N and N-1. An N-N checkpoint is one in which each of a certain number of processes (N) writes to a unique file, for an equal number of files written (N processes = N files written). An N-1 checkpoint differs

in that all N processes write to a single shared file. Although both N-N and N-1 patterns pose challenges, we have observed that the challenges of N-1 checkpointing, from the perspective of the parallel storage system, are much more difficult. Applications using N-1 patterns consistently achieve significantly less bandwidth than do those using an N-N pattern. Because N-N checkpointing derives higher bandwidth than N-1, the path to faster checkpointing is for application developers to rewrite existing N-1checkpointing applications to perform N-N checkpointing instead. Additionally, all new applications should be written to take advantage of the higher bandwidth available to N-N checkpointing. Some developers have gone to N-N checkpointing, but many continue to prefer an N-1 pattern even though its disadvantages are well understood.

Of the 23 applications listed on the Parallel I/O Benchmarks, an industry standard developed by the Parallel I/O Benchmarking Consortium, at least 10 have an N-1 pattern. Two major applications at LANL use an N-1 checkpointing pattern, as do at least two of the eight initial applications chosen to run the Laboratory's Roadrunner, the world's first petascale computer. The parallel storage system attached to Roadrunner is the largest the Laboratory has ever had; testing it has revealed that the challenges of N-1 patterns are severely exacerbated at this scale. Given current bandwidths, we know of no current N-1 application at LANL that can effectively checkpoint across the full width of Roadrunner. PLFS, currently mounted on Roadrunner and many other Laboratory systems, allows them to do so (Fig 2). More generally, PLFS achieves this improvement for any parallel writes and not just checkpoint writes. PLFS also has been shown to improve bandwidth for many parallel reads.

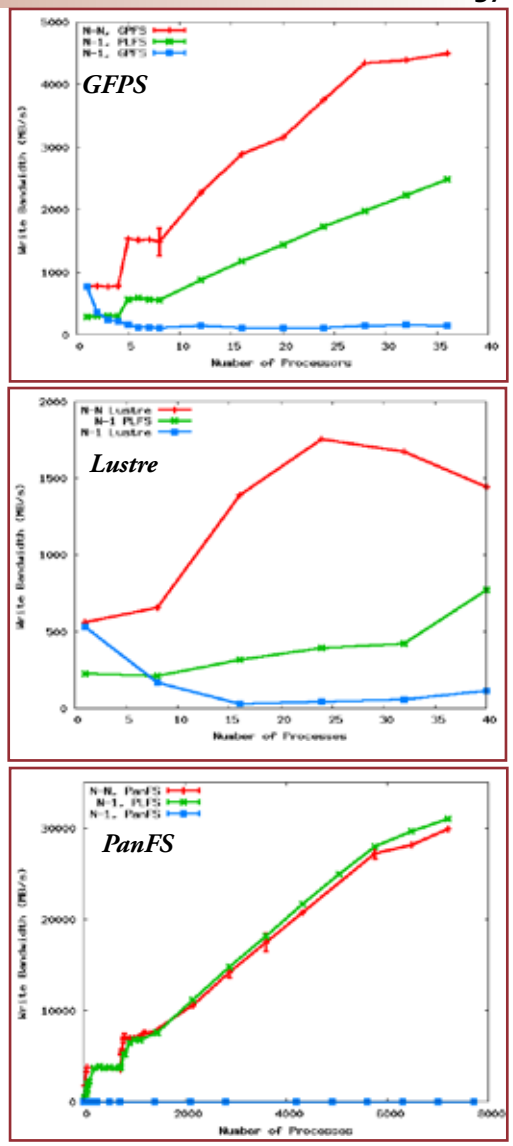
The theory underlying this work is that concurrent, random-access checkpoint writes to a shared file require unnecessary disk-seeking and file locking at the parallel storage system, thereby drastically reducing bandwidth. By decoupling concurrent access and by reorganizing random logical writes into sequential physical writes, PLFS enables applications to write in parallel at a near-optimal storage bandwidth.

In addition to observing a large performance improvement at the Laboratory, which uses the Panasas ActiveScale (PanFS) storage system, we have also tested PLFS on the two other most widely used storage systems for supercomputing: the Linux cluster storage system (Lustre) and GPFS, the IBM General Parallel File system. We have demonstrated a 25X speedup on GPFS, an 8X speedup on Lustre, and a 3000X speedup on PanFS. Notice that the Lustre and GPFS experiments were run on very small clusters. Extrapolating the trends and comparing with the PanFS results suggests that speedups for GPFS and Lustre on reasonably sized clusters will be similar to the PanFS results.

Figure 3 shows the impact of PLFS on three different parallel storage systems: GPFS, Lustre, and Panasas (PanFS). In each graph, the *y*-axis shows the write bandwidth controlled by the number of concurrent writers on the x-axis. The red lines show the bandwidth for N-N workloads, and the blue lines are the bandwidth for small, strided N-1 workloads (*strided* being a type of regular access pattern).

Notice that all three storage systems show massive performance degradation for these N-1 workloads. The green lines are the performance of these same N-1 workloads as written through PLFS. PLFS allows the application to use its preferred N-1 workload and achieve close to the bandwidth it would have achieved if using an N-N workload. For Lustre and GPFS, the N-1 bandwidth through PLFS is much better than the N-1 bandwidth directly to the storage system, but it is not as good as the N-N bandwidth.

For Panasas, however, N-1 through PLFS does match the N-N bandwidth. Notice that the x-axes are very different. The GPFS and Lustre results were gathered on small clusters of fewer than 50 nodes, whereas the Panasas results come from a cluster of several thousand nodes. We expect that the PLFS results on Lustre and GPFS will match N-N bandwidths for larger-sized clusters.



For more information contact John Bent at johnbent@lanl.gov

Funding Acknowledgments

- National Science Foundation (NSF)
- DOE, NNSA Advanced Simulation and Computing (ASC) Program
- DOE, Scientific Discovery through Advanced Computing Program (SciDAC)